TITLE:

Efficient Mini-batch Training on Memristor Neural Network Integrating Gradient Calculation and Weight Update

AUTHOR(S):

YAMAMORI, Satoshi; HIROMOTO, Masayuki; SATO, Takashi

CITATION:

YAMAMORI, Satoshi ...[et al]. Efficient Mini-batch Training on Memristor Neural Network Integrating Gradient Calculation and Weight Update. IEICE Transactions of Fundamentals on Electronics, Communications and Computer Sciences 2018, E101-A(7): 1092-1100

ISSUE DATE:

2018-07-01

URL:

http://hdl.handle.net/2433/242229

RIGHT:

© 2018 The Institute of Electronics, Information and Communication Engineers 許諾条件に基づいて掲載しています。

京都大学学術情報リポジトリ
KURENAI 紅
Kyoto University Research Information Repository

京都大学 KYOTO UNIVERSITY
A Self-archived copy in
Kyoto University Research Information Repository
https://repository.kulib.kyoto-u.ac.jp
京都大学学術情報リポジトリ KURENAI 紅 Kyoto University Research Information Repository

IEICE TRANS. FUNDAMENTALS, VOL.E101–A, NO.7 JULY 2018

PAPER

Efficient Mini-Batch Training on Memristor Neural Network Integrating Gradient Calculation and Weight Update

Satoshi YAMAMORI[†a)], ***Student Member***, **Masayuki HIROMOTO**[†], ***and*** **Takashi SATO**[†], ***Members***

SUMMARY We propose an efficient training method for memristor neural networks. The proposed method is suitable for the mini-batch-based training, which is a common technique for various neural networks. By integrating the two processes of gradient calculation in the backpropagation algorithm and weight update in the write operation to the memristors, the proposed method accelerates the training process and also eliminates the external computing resources required in the existing method, such as multipliers and memories. Through numerical experiments, we demonstrated that the proposed method achieves twice faster convergence of the training process than the existing method, while retaining the same level of the accuracy for the classification results.

key words: *memoristor, neural network, mini-batch training, stochastic gradient descent*

1. Introduction

Resent progress in neural networks is remarkable. The performance of the neural networks have become comparable, or have outperformed the human's ability in various fields such as image recognition, speech recognition, and automatic translation [1]–[4]. This progress owes greatly to the progress of computer technologies along with the Moore's law. However, recent processors, such as CPUs and GPUs, are facing a serious obstacle — the computation in state-of-the-art neural network architectures demand large amount of memory access for input data and network parameters.

A memristor neural network (MNN) [5], [6] is one of the emerging computing technologies to resolve the above problem. A memristor [7]–[9], which is also called as a ReRAM, is a passive electric element whose resistance is changed by the current passing through the device, and can be used as a low-power non-volatile memory [6], [10]. By employing a crossbar array structure, the memristors are also capable of performing multiply-and-accumulate (MAC) operations [5], [11]–[13]. This enables so-called in-memory calculation, which has a possibility to overcome the Von Neumann bottleneck of the modern processors. There are various kinds of MNNs proposed [13]–[16], most of which are reported to realize much better energy efficiency than the conventional digital processing.

The training of the MNNs is realized by applying high voltages to the memristors to update conductance according to the parameters of the neural networks. There are two approaches to train the MNN: ex-situ and in-situ methods [17]. In the ex-situ method, the parameters, or *weights*, of the neural networks are calculated separately from the MNN by executing a training algorithm on ordinary digital computers. This method is so simple that it is used in various existing works [11], [17]. However, it is difficult to accurately write the weights to the memristors due to device variation. Since the error in the weights may deteriorate the classification performance of the neural networks, additional adjustment of the memristor resistance has been necessary. On the other hand, the in-situ method performs network training on the MNN itself. Although this method requires additional peripheral circuits dedicated for training, this method has advantages in terms of robustness against the device variation and capability to realize on-line training.

Kataeva et al. [18] proposed an in-situ training method for the MNN, in which the backpropagation algorithm is efficiently combined with the weight update process on the MNN. This method realized a good training result on the MNN having variations of the memristor devices, as reported by Prezioso et al. [13]. However, this method is not suitable for applying to the algorithms that use a mini-batch technique. The mini-batch-based training is known as fast and robust, and it is used in combination with various optimization algorithms such as a stochastic gradient decent (SGD) method [19]. The reason is that the Kataeva's method requires not only a large number of update iterations in proportion to the size of the MNN, and also additional computation resources, such as multipliers and memories, are required to calculate and store the intermediate data.

In this paper, we propose weight dividing update (WDU) method, an efficient in-situ training method for MNNs with mini-batch technique. The proposed method divides the weight update into discrete processes for each sample in a mini-batch. As a result, we can combine two processes into one: gradient calculation in the backpropagation algorithm, and weight update in the write operation of the memristor. This integration can eliminate the external multipliers and memories that have been indispensable for the existing method, and also can shorten the training process by parallel execution of the weight update for all the memristors on the crossbar array.

The contribution of this paper is summarized as follows.

- A weight dividing update (WDU) method for efficient mini-batch training on memristor neural networks

Manuscript received October 31, 2017.
Manuscript revised March 7, 2018.

[†]The authors are with Department of Communications and Computer Engineering, School of Informatics, Kyoto University, Kyoto-shi, 606-8501 Japan.
a) E-mail: paper@easter.kuee.kyoto-u.ac.jp

DOI: 10.1587/transfun.E101.A.1092

Copyright © 2018 The Institute of Electronics, Information and Communication Engineers

京都大学 KYOTO UNIVERSITY
A Self-archived copy in
Kyoto University Research Information Repository
https://repository.kulib.kyoto-u.ac.jp
京都大学学術情報リポジトリ KURENAI 紅 Kyoto University Research Information Repository

Algorithm 1 Stochastic gradient decent (SGD)

1: Choose an initial weight matrix $\boldsymbol{W}$
2: **for** $i = 1, \ldots, N_{\text{epochs}}$ **do**
3: Randomly sample a mini-batch $\mathcal{B}$ from the training set
4: Calculate gradient $\Delta \boldsymbol{W}$ on the mini batch $\mathcal{B}$
5: Update weight: $\boldsymbol{W} \leftarrow \boldsymbol{W} + \Delta \boldsymbol{W}$
6: **end for**

(MNNs) is newly proposed.

- WDU eliminates the external multipliers and reduces memory usage compared to the existing method [18].
- WDU can accelerate the training process by parallel weight update of the memristors.
- The training accuracy is evaluated through circuit simulation. The result shows the proposed WDU achieves M/K times faster convergence of the training compared to the existing method [18], while keeping the same classification accuracy. Here, M and K are a network size and a mini-batch size, respectively. The mini-batch is a subset of the training dataset, from which the gradient is calculated for SGD to update weights of the neural network. The mini-batch size K is the number of the data elements in the mini-batch.

The remainder of the paper is organized as follows. First, the fundamentals of conventional MNNs are briefly reviewed in Sect. 2. Then the proposed WDU method is described in Sect. 3, and the performance evaluation through circuit simulation is conducted Sect. 4. Finally, the paper is concluded in Sect. 5.

2. Memristor Neural Network

2.1 Neural Network

A neural network is one of the brain-inspired mathematical models that have an ability to approximate functions through learning on a large training data [19]. The neural network has a hierarchical network structure that consists of simple units called neurons. Figure 1 shows a neuron model and an example of the multi-layer neural network. The output of the neuron is given by

$$\boldsymbol{y} = f(\boldsymbol{W}\boldsymbol{z}), \tag{1}$$

where $\boldsymbol{z}$, $\boldsymbol{W}$, and $f(\cdot)$ are an input vector, a weight matrix, and an activation function, respectively.

In training the neural networks, stochastic gradient decent (SGD) [19] is widely used. Algorithm 1 shows a pseudocode of the SGD, in which gradient calculation (Line 4) and weight update (Line 5) are iteratively executed for each epoch until the iteration reaches a predefined number of epochs, N_{epochs}. The weight gradient $\Delta \boldsymbol{W}$ is obtained by the backpropagation method as follows:

$$\boldsymbol{e}^L(k) = \boldsymbol{t}(k) - \boldsymbol{y}(k), \tag{2}$$

$$\boldsymbol{e}^l(k) = \left(\left(\boldsymbol{W}^{l+1}\right)^{\mathrm{T}} \boldsymbol{e}^{l+1}(k)\right) \odot f'\left(\boldsymbol{W}^l \boldsymbol{z}^l(k)\right), \tag{3}$$

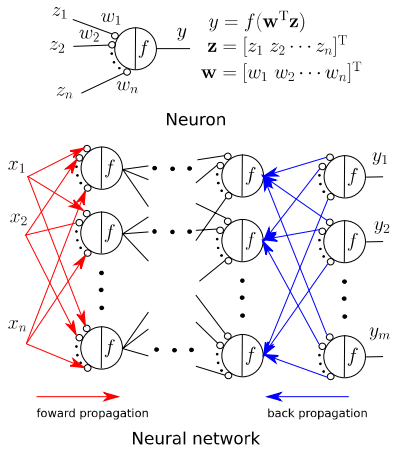


Fig. 1 A neuron model and an example of a multi-layer neural network. A neuron works as a scalar function that calculates an inner product of the input $\boldsymbol{z}$ and the weight $\boldsymbol{w}$ and outputs $\boldsymbol{y}$ through an activation function $f(\cdot)$. The neural network consists of multiple neurons connected to each other. The red and blue arrows represent the directions of the forward and back propagation, respectively.

$$\Delta^l(k) = \boldsymbol{e}^l(k)\left(\boldsymbol{z}^l\right)^{\mathrm{T}}(k), \tag{4}$$

$$\Delta \boldsymbol{W}^l = \eta \frac{1}{K} \sum_{k=1}^{K} \Delta^l(k). \tag{5}$$

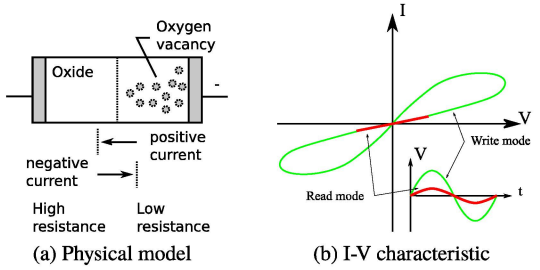
Here we assume an L-layer neural network, and $\boldsymbol{z}^l$, $\boldsymbol{W}^l$, and $\boldsymbol{e}^l$ are the input, weights, and errors for the l-th layer, respectively. $\boldsymbol{y}$ is the output of the final layer, i.e., the result of the forward propagation of the neural network. $\boldsymbol{t}$ is target values of the output, and the error at the final layer $\boldsymbol{e}^L$ is calculated by Eq. (2). Note that the augment k for each variable indicates that those values correspond the k-th training sample in a mini-batch $\mathcal{B}$ whose size is K. The errors at the l-th layer can be obtained from the errors at the $(l+1)$-th layer as shown in Eq. (3). The operator symbols $\odot$, T, and $'$ mean Hadamard product, matrix transpose, and derivative, respectively. By repeating this process, errors for all the layers are obtained sequentially from the last layer to the first layer. This is why this method is called backpropagation. After having all the errors $\boldsymbol{e}^l$, the gradient of the weights to be updated is calculated by Eqs. (4) and (5), where η is a constant variable called a learning rate.

2.2 Memristor Neural Network

2.2.1 Memristor Device

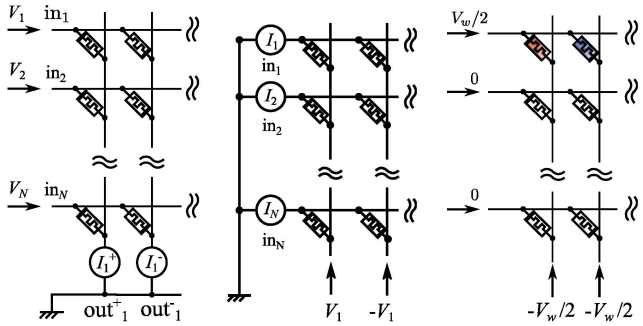
A memristor is a passive two-terminal component theoretically predicted by Chua in 1970s [7] and found by Strukov et al. in 2008 [9]. Figure 2(a) shows a typical structure of the memristor, which consists of a metal oxide layer in between two metal layers. The conductance of the memristor changes depending on the current passing through the device. The I-V characteristic of the memristor is shown in Fig. 2(b), in which the hysteresis loop is found. The memristor works in two operation modes: read and write. In the read mode, a

京都大学 KYOTO UNIVERSITY
A Self-archived copy in
Kyoto University Research Information Repository
https://repository.kulib.kyoto-u.ac.jp
京都大学学術情報リポジトリ KURENAI 紅 Kyoto University Research Information Repository



(a) Physical model (b) I-V characteristic

Fig. 2 The physical abstraction model and the I-V characteristic of the memristor device.



(a) Foward propagation (b) Back propagation (c) Weight update

Fig. 3 Schematics of the memristor crossbar array operating in three modes: (a) forward propagation, (b) back propagation, and (c) weight update by V/2 scheme.

low read voltage V_r is applied to the memristor so that its conductance does not change and the device behaves as a constant conductance. The read current is given by

$$\text{read}: \quad I = GV_\mathrm{r}, \tag{6}$$

where G is the conductance of the memristor. On the other hand, in the write mode, the conductance of the memristor is altered by applying a high write voltage V_w for a certain period, ΔT. The relative change of the conductance is given by

$$\text{write}: \quad \frac{\Delta G}{G} = \delta_G \propto \Delta T \exp(\alpha V_\mathrm{w}), \tag{7}$$

where α is a device-dependent constant [20].

2.2.2 Crossbar Array

A crossbar array is a typical circuit structure to implement neural networks using memristors [12]. Figure 3 shows an example of the memristor crossbar array, in which the memristors are connected at each intersection of the horizontal input line and the vertical output line. A pair of the memristors is used to represent the positive and negative weights of the neural network. The crossbar array works in three modes: forward propagation, back propagation, and weight update.

First, the forward propagation mode is explained using Fig. 3(a). Let the conductances of the positive and negative memristors at the intersection of the i-th column and the j-th row be G^+_{ij} and G^-_{ij} ($i = 1, \cdots, M$, $j = 1, \cdots, N$), respectively. When voltages V_j are applied to the input terminals in_j, the currents flowing through the output terminals out^+_i and out^-_i become

$$I^+_i = \sum_j^N G^+_{ij} V_j, \quad I^-_i = \sum_j^N G^-_{ij} V_j. \tag{8}$$

In matrix representation, this operation can be written as

$$\boldsymbol{I}_\mathrm{fwd} = \boldsymbol{G}\boldsymbol{V}_\mathrm{fwd}, \tag{9}$$

where $\boldsymbol{I}_\mathrm{fwd}$ is a vector of differential output currents $I_i = I^+_i - I^-_i$, and $\boldsymbol{V}_\mathrm{fwd}$ is a vector of the input voltages V_j. Each element in the conductance matrix $\boldsymbol{G}$ represents the difference of the pairwise conductances, $G_{ij} = G^+_{ij} - G^-_{ij}$. When we associate $\boldsymbol{G}$ and $\boldsymbol{V}_\mathrm{fwd}$ with the weights $\boldsymbol{W}$ and the inputs $\boldsymbol{z}$ of neurons in Eq. (1) respectively, the crossbar array realizes the dot product operations to obtain the output $\boldsymbol{y}$ as the output currents $\boldsymbol{I}_\mathrm{row}$. As such, in the crossbar array, the forward propagation of the neural network is conducted by only using the passive components without using costly digital multipliers.

Next, the backpropagation is also realized as shown in Fig. 3(b). Compared to the forward propagation mode, the inputs and the outputs of the crossbar array are swapped. By applying voltages $\boldsymbol{V}_\mathrm{bwd}$ to the bottom terminals (which were the output terminals in the forward mode) of the corssbar array, the currents flown through the left terminals (input terminals in the forward mode) become

$$\boldsymbol{I}_\mathrm{bwd} = \boldsymbol{G}^\mathrm{T}\boldsymbol{V}_\mathrm{bwd}, \tag{10}$$

which can realize the dot product operations of $\boldsymbol{W}^\mathrm{T}_{l+1}\boldsymbol{e}_{l+1}(k)$ in Eq. (3), by associating $\boldsymbol{G}$ and $\boldsymbol{V}_\mathrm{bwd}$ with $\boldsymbol{W}_{l+1}$ and $\boldsymbol{e}_{l+1}(k)$, respectively.

Finally, the weight update mode is explained. In this paper, "V/2 scheme" [6], [13] is used to update the conductance of the memristor in the corssbar array. This method enables to change the conductance of only the selected memristor while retaining those of the other memristors. Figure 3(c) shows an example of the voltage application to update the conductance G^+_{00} and G^-_{00} only. The terminals connected to the target memristors are set to $V_\mathrm{w}/2$ and $-V_\mathrm{w}/2$, while the other terminals are set to 0 V. Here, V_w is a write voltage that is determined to satisfy

$$V_\mathrm{w}/2 < V_\mathrm{th} < V_\mathrm{w}, \tag{11}$$

where V_th is a threshold voltage under which the conductance of the memristor is almost unchanged. When the voltages are applied, the conductances of G^+_{00} and G^-_{00} are changed since the applied voltage V_w is higher than the threshold V_th, whereas the conductances of the other memristors remain same since the applied voltage does not reach the threshold voltage. In this way, by controlling the voltage to apply to the crossbar array, only the desired memristors can be updated. The determination of the write voltage V_w and the efficient

京都大学 KYOTO UNIVERSITY
A Self-archived copy in
Kyoto University Research Information Repository
https://repository.kulib.kyoto-u.ac.jp
京都大学学術情報リポジトリ KURENAI 紅 Kyoto University Research Information Repository

weight update method on the crossbar array are detailed in the next section.

2.2.3 Variable-Amplitude Scheme

In this section, we describe the "variable-amplitude" scheme proposed by Kataeva et al. [18] for in-situ training of the neural networks on the memristor crossbar array. From Eq. (7), the conductance change $\Delta \boldsymbol{G}$ is determined by the period ΔT and the amplitude V_{w} of the voltage application. In the variable-amplitude scheme, the write voltage V_{w} is changed to control the conductance, while the period ΔT is kept constant. The advantage of this method is that it can substitute multiplication of two variables with subtraction of two voltages applied to the memristor. For example, assume that we want to update the conductance by $\Delta G \propto XY$, where X and Y are two variables whose product is proportional to ΔG and satisfying $|X|, |Y| > 1$. Let the two voltages V_x and V_y be

$$V_x = \mathrm{Sign}(XY) \ln |X|, \tag{12}$$
$$V_y = -\mathrm{Sign}(XY) \ln |Y|. \tag{13}$$

When these two voltages are applied to the two terminals of the memristor, its conductance changes by

$$\Delta G \propto \exp(V_{\mathrm{w}}) \tag{14}$$
$$= \exp(V_x - V_y) \tag{15}$$
$$= \exp(\mathrm{Sign}(XY) \ln |XY|) \tag{16}$$
$$= XY. \tag{17}$$

This means that the multiplication of X and Y can be realized just by applying corresponding voltages V_x and V_y.

This property of the variable-amplitude scheme enables effective in-situ training on the memristor crossbar array when not using a mini-batch, i.e., $K = 1$. In this case, the conductance of the memristor at the intersection of the i-th row and the j-th column should be updated by $\Delta G_{ij} \propto \Delta W_{ij} = \eta e_i z_j$ according to Eq. (5). This update is realized in a fully parallel manner on the crossbar array by applying voltages $V_j \propto \ln |z_j|$ to the j-th row and $V_i \propto \ln |e_i|$ to the i-th column respectively as shown in Fig. 4. Note that V_i and V_j must be chosen to satisfy the V/2 scheme condition in Eq. (11). Since the signs of the two values z_j and e_i have four combinations of $(+,+)$, $(+,-)$, $(-,-)$, and $(-,+)$, the update process must be separately executed in four phases corresponding to such sign combinations, which is detailed in [18]. Thus the number of the voltage application for each weight update becomes $N_{\mathrm{va}} = 4$. The advantage of this method is that it does not require external circuits to calculate the multiplication of $z_j \times e_i$, but requires $N + M$ memories to store z_j and e_i during the backpropagation process.

In the case of the mini-batch training with $K > 1$, however, the above method does not work efficiently. From Eq. (5), the conductance update of the memristor should be $\Delta G_{ij} \propto \Delta W_{ij} = \eta \frac{1}{K} \sum_{k=1}^{K} e_i(k) z_j(k)$, which cannot be decomposed into a simple multiplication of two variables, such as $e_i(k)$ and $z_j(k)$. Therefore, we must calculate ΔW_{ij}

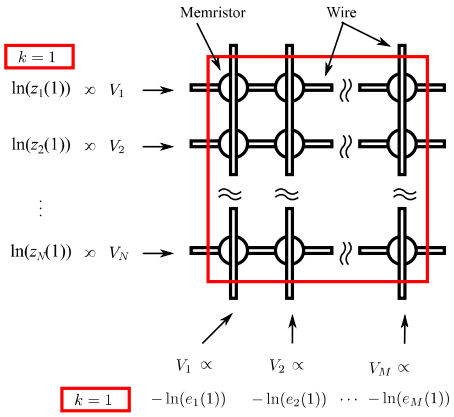


Fig. 4 Weight update by existing method with $K = 1$.

Algorithm 2 Weight update by existing method with $K > 1$

```
K ← mini batch size
M ← row sizes
k ← 1
while k ≤ K do
    Do forward propagation
    Store the input data z_k^l for each layer l
    Do back propagation
    Store the update value ΔW_k^l for each layer l
    k ← k + 1
end while
m ← 1
while m ≤ M do
    Update weights in the m-th column by ΔW_k^l
    m ← m + 1
end while
```

off the crossbar, and apply voltages $V_j \propto \ln |\Delta W_{ij}/C|$ to the j-th row and $V_i \propto \ln(C)$ to the i-th column, respectively. Note that $C > 0$ is a constant such that V_i and V_j satisfy the V/2 scheme condition in Eq. (11). Because the values of V_j are different for each i-th column, the update process requires M iterations as shown in Alg. 2 and Fig. 5. Each iteration is executed in two phases since C is a positive constant and only the sign of ΔW_{ij} must be considered. Thus, the total number of the voltage application becomes $N_{\mathrm{va}} = 2M$. This means that the time required for the weight update proportionally increases as the size of the crossbar becomes larger. In addition, unlike the case of $K = 1$, this method requires KMN multiplications and MN memories for the external circuit to calculate and store ΔW_{ij}, which may deteriorate overall efficiency of the system that is based on the MNN.

3. Weight Dividing Update

In this section, we propose a novel weight update method called "weight dividing update (WDU)," which realizes efficient mini-batch training on the memristor crossbar array. As described in the previous section, the existing method [18] does not work efficiently for the mini-batch training since the multiplication and the summation for the K samples in Eqs. (4) and (5) must be calculated in the external system. In contrast, our proposed method performs this calculation

京都大学 KYOTO UNIVERSITY
A Self-archived copy in
Kyoto University Research Information Repository
https://repository.kulib.kyoto-u.ac.jp
京都大学学術情報リポジトリ KURENAI 紅 Kyoto University Research Information Repository

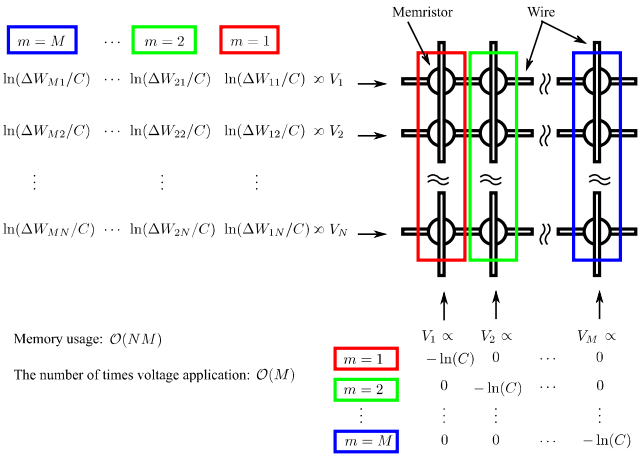


Fig. 5 Weight update by the conventional method when $K > 1$. One column in each iteration is selected to apply column voltage, and the memristors in the selected column are simultaneously updated.

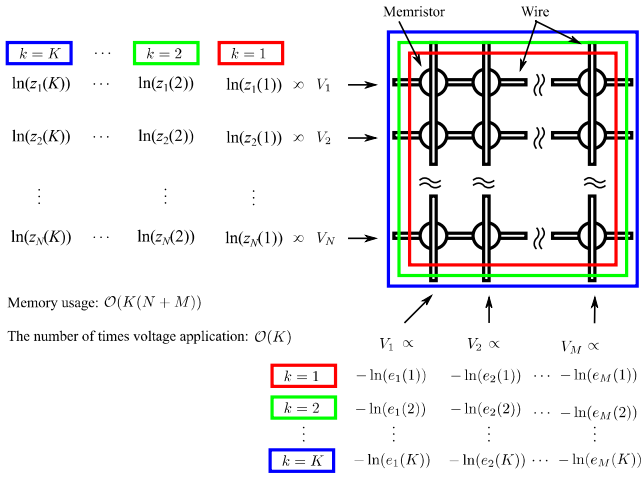


Fig. 6 Weight update by the proposed method. The memristors within a colored rectangle are updated in parallel for each iteration.

Algorithm 3 Weight dividing update method

$K \leftarrow$ mini batch size
$k \leftarrow 1$
while $k \leq K$ **do**
  Do forward propagation
  Store the input data $\boldsymbol{z}_k^l$ for each layer l
  Do back propagation
  Store the errors $\boldsymbol{e}_k^l$ for each layer l
  $k \leftarrow k + 1$
end while
$k \leftarrow 1$
while $k \leq K$ **do**
  Update weights using the input data $\boldsymbol{z}_k^l$ and the errors $\boldsymbol{e}_k^l$
  $k \leftarrow k + 1$
end while

inside the crossbar array in addition to the weight update.

Algorithm 3 shows the procedure of the weight update by the proposed method. First, the input data $\boldsymbol{z}(k)$ and the corresponding errors $\boldsymbol{e}(k)$ are calculated through forward and backward propagation for each training sample k in the mini-batch. Then the weight update is executed for each k, without calculating the summation of K samples in Eq. (5). In other words, the weight ΔW is *divided* into K values of $z_j(k) \cdot e_i(k)$ and they are sequentially updated through K iterations. This method brings another great advantage that the multiplications of $z_j(k)$ and $e_i(k)$ can be realized on the crossbar array in a fully parallel manner, as in the case of the existing method [18] with $K = 1$. Figure 6 illustrates the update flow of the proposed method, in which, for each iteration, all the memristors are updated in parallel. Note that each update is identical to that of the existing method [18] with $K = 1$, and it requires four phases due to the sign combination of the values. Thus the total number of the voltage application is $N_{\mathrm{va}} = 4K$ for the proposed WDU.

Table 1 summarizes the comparison of the existing methods [18] and the proposed method. The number of the voltage application of WDU is $N_{\mathrm{va}} = 4K$ whereas that of the existing method with $K > 1$ is $N_{\mathrm{va}} = 2M$. This means that WDU outperforms the existing methods when $K < M/2$, which is usually the case in recent practical deep neural networks. Here, we define a metric called required clocks per epoch (CPE), which measures efficiency of setting weights:

$$\mathrm{CPE} = C_{\mathrm{fwd}} \times D + C_{\mathrm{bwd}} \times D + C_{\mathrm{upd}} \times D/K, \qquad (18)$$

where D is a total number of the training data, and K is a mini-batch size. C_{fwd} and C_{bwd} are the numbers of required clocks to execute the forward and backward propagation for a single training data, respectively. In general, they are constant values. C_{upd} is a number of required clocks for weight update, i.e., the number of voltage application, N_{va}. From this definition, the required clocks per data (CPE/D) can be calculated as shown in Table 1. The order of the CPE/D for the proposed method is $\mathcal{O}(1)$, which is independent of the size of the crossbar array. Another advantage of WDU is smaller footprint of external memories than the existing method, when the crossbar size M is sufficiently larger than the batch size K. In addition, WDU requires no multipliers for all M and K, while existing method requires KMN multiplires when $K > 1$.

Table 1 Comparison of the existing methods and proposed method WDU.

	Existing methods [18]		WDU
	($K = 1$)	($K > 1$)	(All K)
# Voltage applications, N_{va}	4	$2M$	$4K$
Clocks per data, CPE/D	$\mathcal{O}(1)$	$\mathcal{O}(M/K)$	$\mathcal{O}(1)$
External memory	$M + N$	MN	$K(M + N)$
External multipliers	0	KMN	0
# Voltage sources for Forward	M	M	M
# Voltage sources for Back	N	N	N
# Voltage sources for Update	$M + N$	$N + 1$	$M + N$

Table 1 also shows the number of voltage sources that are connected to the row/column terminals. In the weight update mode, the proposed method requires $M + N$ voltage sources whereas the existing method for $K > 1$ requires only $N + 1$. This means that the circuit area of the proposed

京都大学 KYOTO UNIVERSITY
A Self-archived copy in
Kyoto University Research Information Repository
https://repository.kulib.kyoto-u.ac.jp
京都大学学術情報リポジトリ KURENAI 紅 Kyoto University Research Information Repository

method becomes larger than that of the existing method since each voltage source is typically realized by a digital-to-analog converter (DAC). However, as the size of the crossbar (M and N) becomes larger, the area of the DACs becomes relatively smaller than that of the crossbar array since the former grows linearly to M and N, while the latter grows in the order of $O(MN)$. Therefore, the increase of the circuit area is remedied when we assume to use a sufficiently large crossbar array as in our research target. In addition, the additional DACs required for the proposed method does not deteriorate the operation frequency, since all the DACs operate in parallel and thus their conversion time is equal to the single DAC.

The training results of the MNNs by WDU and the existing method are basically identical since both methods are based on the same equation to determine the update weight as Eq. (5). However, this claim is true under the assumption that the conductance of the memristor G in Eq. (7) is constant. In reality, the conductance G changes gradually during the WDU process, which may cause difference between the final results after training of the two methods. In this work, however, we assume that the conductance change is negligibly small and does not affect the training process. This assumption is validated through the experiments in the next section, which shows that the training results by the proposed method achieved almost equal accuracy to the existing method.

4. Experiments

In this section, through circuit simulations using a memristor device model, we demonstrate that the proposed WDU method can achieve equal accuracy with the conventional method in training MNNs.

4.1 Memristor Model

In the following experiments, we use a memristor model by Chen et al. [21], which is a behavioral model written in Verilog-A language and for analog circuit simulation using SPICE. The model parameters in this model are used, which are based on the measurements on a HfO_x-based ReRAM. The I-V characteristic of the memristor model follows Eq. (7) in the region where δ_G is small. The write voltage to change the conductance of the memristor by δ_G can be determined as

$$V_w = A \ln(\kappa \delta_G) + B, \tag{19}$$

where $A = 0.03864$, $B = 2.030$, and $\kappa = 0.05$ are the constants calculated from the model parameters for the voltage application time of $\Delta T = 3.5$ ns. Note that the signs of V_w and δ_G do not match if $\delta_G < \exp(-B/A)/\kappa < 2.1 \times 10^{-22}$, but this case rarely occurs in a practical use.

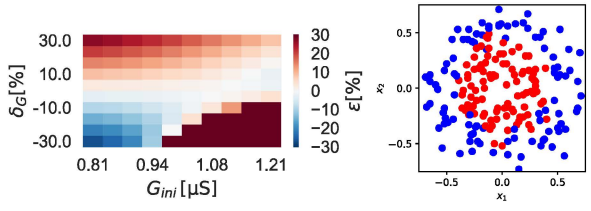
On the other hand, if δ_G takes a large value, Eq. (7) does not hold, and the variable-amplitude method is not applicable to train the MNN. Therefore, we must use the memristor device under the write voltage V_w such that δ_G is maintained small.

In order to determine an adequate V_w, we first conduct a preliminary experiment for a single memristor by SPICE simulation. Given a memristor having a certain initial conductance G_{ini}, a write voltage V_w, which is determined by Eq. (19) to increase the conductance by δ_G, is applied, and the error $\varepsilon = G - G_{ini}(1+\delta_G)$ is observed, where G is a new conductance after the application of the write voltage. Figure 7(a) shows conductance error ε for various combinations of G_{ini} (x-axis) and δ_G (y-axis). Note that δ_G is presented as a relative percentage against G_{ini}. From the result, the error is suppressed fewer than 10% if the conductance change δ_G is less than 10%, whereas the intolerable errors are found in the region with large δ_G. Thus the training parameters are determined so that δ_G becomes less than 10% of the initial conductance.

4.2 Simulation Setups

For a benchmark to evaluate the memristor neural network, a binary classification problem called "circle" [22] is utilized. The classification task is to separate the two-dimension input vectors (x_1, x_2) into two classes. An example input vectors are shown in Fig. 7(b), in which the red and blue present the members of the two classes. All the input vectors are distributed in the rage of $[-1.0, 1.0]$. Among 200 samples in the dataset, we utilize 150 samples for training and 50 samples for testing, respectively.

The configuration of the neural network to be implemented on the memristor crossbar array is a two-layer fully connected network with a 2-M first layer and M-1 second layer. The numbers of the input and output channels are one and two, respectively. The total number of the memristors to be used is $((2+1)\times M + (M+1)\times 1)\times 2 = 8M+2$. Note that "+1" is for the bias term, and "×2" is for the pair of memristors to represent positive and negative weights as described in Sect. 2.2.2. The initial conductances of the memristors G_{ini} are determined by changing the model parameter g, a gap, which is a length of high-impedance region in a memristor device. For each memristor, g is sampled from a normal distribution with the mean of 1.37 nm and the relative variance of 0.01. For the activation functions, a hyperbolic tangent



(a) Errors of a single memristor. (b) "circle."

Fig. 7 (a) Errors of conductance change of a single memristor. G_{ini} is an initial conductance and δ_G is an increase rate of the target conductance. ε is a relative error of the simulation result against the ideal value from Eq. (7). (b) An example binary classification problem "circle." The red and blue points (x_1, x_2) are the samples belonging to two classes. They are distributed within the range of $[-1.0, 1.0]$.

京都大学 KYOTO UNIVERSITY
A Self-archived copy in
Kyoto University Research Information Repository
https://repository.kulib.kyoto-u.ac.jp
京都大学学術情報リポジトリ KURENAI 紅 Kyoto University Research Information Repository

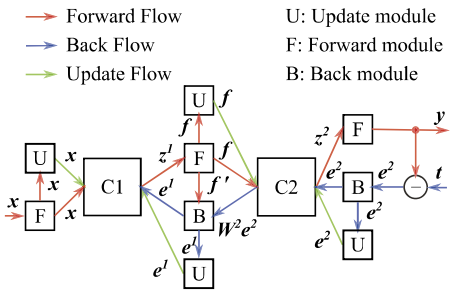


Fig. 8 Simulation circuit for training of the two-layer memristor neural network.

Table 2 Applied voltage for weight update.

Applied voltage	Existing method [18]	Proposed method
$V_w = V_{row} - V_{col}$	$A\ln(\Delta W) + B$	$A\ln(x_i(k)e_j(k)) + B$
Rows: V_{row}	$A\ln(\Delta W/C) + B$	$A\ln(x_i(k)) + B/2$
Columns: V_{col}	$-A\ln(C)$	$-A\ln(e_j(k)) - B/2$

$f(x) = \tanh(\beta x)$ for the first layer, where β is a hyper parameter to normalize the input, and a sigmoid function $f(x) = 1/(1+\exp(-\beta x))$ for the output layer are used.

For circuit simulation, we implement the crossbar array and the other peripheral circuits with Verilog-A language. The memristor model described in Sect. 4.1 is used for the crossbar array, and the peripheral circuits such as current controlled voltage sources (CCVS), activation functions, and a control circuit are implemented as ideal behavioral models. In a practical use, the peripheral circuits are realized by A/D and D/A converters or mixed-signal circuits with similar functions [13]–[16]. Figure 8 shows a diagram of the simulation circuit and its behavior. The three modes are operated on the same circuit: forward propagation (red arrows), back propagation (blue arrows), and weight update (green arrows). "C1" and "C2" are the crossbar arrays for the first and the second layers, respectively. "F," "B," and "U" units are the modules to control the operations for the forward propagation, back propagation, and weight update modes. Each module includes CCVSs to provide the voltage to the corresponding crossbar array and memories to store the data such as z and e. In the forward and back propagation modes, the applied voltage to the memristor is limited to $|V_i| < 0.1$ V so that the conductance of the memristor does not change. In the weight update mode, the applied voltages by the existing method [18] and the proposed method are summarized in Table 2.

The hyper parameters used for the training are experimentally determined by a grid search. We set the learning rate to $\eta = 0.01$ and the normalization parameter for tanh activation to $\beta = 1 \times 10^8\ \mathrm{A}^{-1}$.

4.3 Result

The classification accuracies of the existing method [18] and the proposed method are compared with different sets of the parameters K (mini-batch size) and M (crossbar array size). Table 3 shows the classification errors on the test set by the existing and the proposed methods. The result shows that the proposed method achieves comparable or lower error rates compared to the existing method. The minimum error is 6.0%, which is almost equal to the software classification results. Thus the proposed method successfully performs the sufficient level of the neural network training.

Table 3 Classification errors by the existing method [18] / the proposed method (%).

K \ M	4	8	16	32
4	8.0 / 10.0	22.0 / 12.0	8.0 / 8.0	6.0 / 6.0
8	8.0 / 8.0	12.0 / 8.0	6.0 / 6.0	6.0 / 6.0
16	22.0 / 8.0	20.0 / 16.0	6.0 / 6.0	8.0 / 6.0

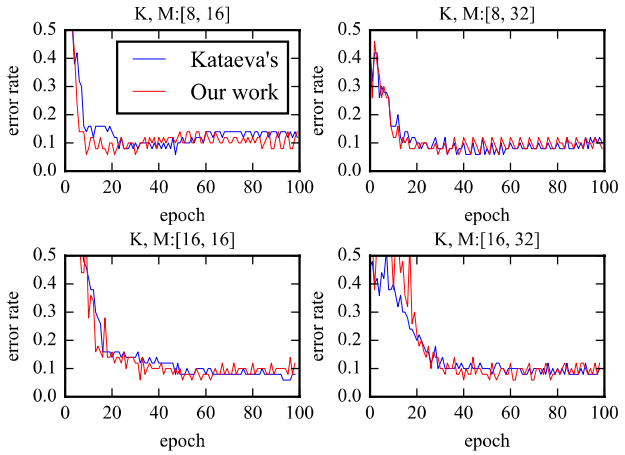


Fig. 9 Learning curves by the existing method and the proposed method. The classification error is shown as function of the number of elapsed epochs.

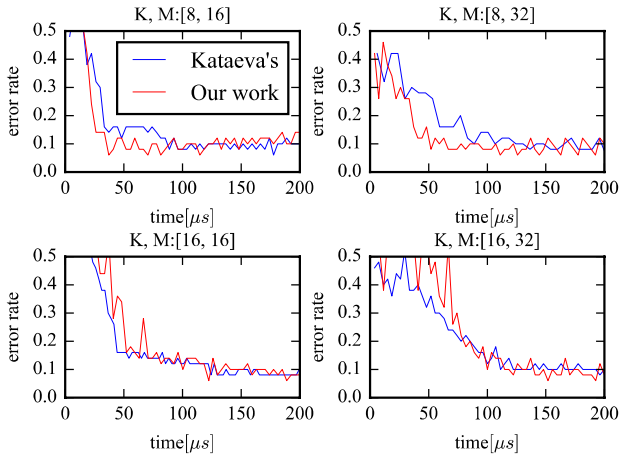


Fig. 10 Learning curves by the existing method and the proposed method. The classification error is shown as function of the elapsed time.

Figure 9 shows the learning curves for $(K, M) = (8, 16)$, $(8, 32)$, $(16, 16)$, $(16, 32)$, which represent time change of the classification error for the test set as a function of training epochs. These results indicate that the proposed method achieves the equivalent training of the neural networks to the existing method. Figure 10 shows the same

京都大学 KYOTO UNIVERSITY
A Self-archived copy in
Kyoto University Research Information Repository
https://repository.kulib.kyoto-u.ac.jp
京都大学学術情報リポジトリ KURENAI 紅 Kyoto University Research Information Repository

Table 4 Clocks per data (CPE/D) by the existing method [18] / the proposed method.

K \ M	4	8	16	32
4	4 / 6	6 / 6	10 / 6	18 / 6
8	3 / 6	4 / 6	6 / 6	10 / 6
16	2.5 / 6	3 / 6	4 / 6	6 / 6

learning curves in Fig. 9 but the x-axis is changed to the elapsed time. The elapsed time is calculated by using CPE/D in Table 4 and the clock period. From Eq. (18), CPE/D for the existing and the proposed methods are $(2 \times M/K + 2)$ and 6, respectively. Here, we designed $C_{\text{fwd}} = C_{\text{bwd}} = 1$ for both methods and $C_{\text{upd}} = 2M$ and 4 for the existing and the proposed methods, respectively. In this experiment, the same clock period of 3.5 ns are used for both methods since their peripheral circuits are almost the same and the maximum delay of both methods is equal. The result when $(K, M) = (8, 32)$, which satisfies the condition of $K < M/2$ described in Sect. 3, shows the twice faster convergence of the proposed method than the existing method. For example, the error rate of the proposed method becomes 0.1 at 50 μs, while that of the existing method reaches the same error rate after spending 100 μs.

5. Conclusion

This paper proposed an efficient mini-batch-based training method for MNNs. In the proposed method, we integrate the two essential processes for the training, i.e., the gradient calculation and the weight update, into one process. As a result, the required number of voltage applications to the memristor crossbar array becomes proportional to the size of the mini-batch, whereas that of the existing method is proportional to the size of the crossbar array. The proposed method is suitable when the large crossbar arrays are used. In addition, the external multipliers and memories required by the existing method are greatly reduced by the proposed method. Through the experiments by circuit simulation, it is shown that the proposed method achieves twice faster training process than the existing method when relatively large networks such as $(K, M) = (8, 32)$ are used, while retaining the same level of the accuracy for the classification results.

For the future work, we will consider the design of the peripheral circuits that are suitable for training of neural networks considering the analog characteristics of memristors. Acceleration of the circuit simulation is another future work so that we can accurately evaluate much larger size of neural networks.

Acknowledgments

This work was partially supported by JSPS KAKENHI Grant No. 26730027 and 17H01713. This work was also supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc.

References

[1] A. Krizhevsky, I. Sutskever, and G.E. Hinton, "ImageNet classification with deep convolutional neural networks," Proc. Neural Information Processing Systems, pp.1097–1105, 2012.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," Proc. Computer Vision and Pattern Recognition, pp.770–778, 2016.
[3] G. Hinton, L. Deng, D. Yu, G.E. Dahl, A.r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," IEEE Signal Process. Mag., vol.29, no.6, pp.82–97, 2012.
[4] Y. Wu, Mi. Schuster, Z. Chen, Q.V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Ł. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," Computing Research Repository, vol.abs/1609.08144, 2016.
[5] K.K. Likharev, "CrossNets: Neuromorphic hybrid CMOS/nanoelectronic network," Science of Advanced Materials, vol.3, no.3, pp.322–331, June 2011.
[6] J.J. Yang, D.B. Strukov, and D.R. Stewart, "Memristive devices for computing," Nature Nanotechnology, vol.8, pp.13–24, 2013.
[7] L.O. Chua, "Memristor — The missing circuit element," IEEE Trans. Circuit Theory, vol.18, no.5, pp.507–519, Sept. 1971.
[8] L.O. Chua and S.M. Kang, "Memristive devices and systems," Proc. IEEE, vol.64, no.2, pp.209–223, 1976.
[9] D.B. Strukov, G.S. Snider, D.R. Stewart, and R.S. Williams, "The missing memristor found," Nature, vol.453, no.7191, pp.80–83, 2008.
[10] H.S.P. Wong, H.Y. Lee, S. Yu, Y.S. Chen, Y. Wu, P.S. Chen, B. Lee, F.T. Chen, and M.J. Tsai, "Metal-oxide RRAM," Proc. IEEE, vol.100, no.6, pp.1951–1970, June 2012.
[11] F. Alibart, L. Gao, B.D. Hoskins, and D.B. Strukov, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," Nanotechnology, vol.23, no.7, p.075201, 2012.
[12] M. Hu, H. Li, Y. Chen, Q. Wu, G.S. Rose, and R.W. Linderman, "Memristor crossbar-based neuromorphic computing system: A case study," IEEE Trans. Neural Netw. Learning Syst., vol.25, no.10, pp.1864–1878, Oct. 2014.
[13] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. Adam, K.K. Likharev, and D.B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," Nature, vol.521, no.7550, pp.61–64, 2015.
[14] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J.P. Strachan, M. Hu, R.S. Williams, and V. Srikumar, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," ACM/IEEE 43rd ISCA, pp.14–26, June 2016.
[15] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIMES: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," ACM/IEEE 43rd ISCA, pp.27–39, June 2016.
[16] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," IEEE International Symposium on HPCA, pp.541–552, Feb. 2017.
[17] F. Alibart, E. Zamanidoost, and D.B. Strukov, "Pattern classification by memristive crossbar circuits using ex situ and in situ training," Nature Communications, vol.4, p.2072, June 2013.
[18] I. Kataeva, F. Merrikh-Bayat, E. Zamanidoost, and D. Strukov, "Efficient training algorithms for neural networks based on memristive crossbar circuits," Proc. International Joint Conference on Neural Networks, pp.1–8, July 2015.
[19] C.M. Bishop, Pattern Recognition and Machine Learning, Springer,

京都大学 KYOTO UNIVERSITY
A Self-archived copy in
Kyoto University Research Information Repository
https://repository.kulib.kyoto-u.ac.jp
京都大学学術情報リポジトリ KURENAI 紅 Kyoto University Research Information Repository

2006.
[20] F.M. Bayat, B. Hoskins, and D.B. Strukov, "Phenomenological modeling of memristive devices," Appl. Phys. A, vol.118, pp.779–786, March 2015.
[21] P.Y. Chen and S. Yu, "Compact modeling of RRAM devices and its applications in 1T1R and 1S1R array design," IEEE Trans. Electron Devices, vol.62, no.12, pp.4022–4028, Dec. 2015.
[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol.12, pp.2825–2830, 2011.

Satoshi Yamamori recieved B.E. degree in Electrical and Electronic Engineering from Kyoto University in 2017. He is a master course student at Department of Systems Science, Kyoto University. He is a student member of the Institue of Electronics, Information and Comunication Engineers (IEICE).

Masayuki Hiromoto received B.E. degree in Electrical and Electronic Engineering and M.Sc. and Ph.D. degrees in Communications and Computer Engineering from Kyoto University in 2006, 2007, and 2009 respectively. He was a JSPS research fellow from 2009 to 2010, and with Panasonic Corp. from 2010 to 2013. In 2013, he joined the Graduate School of Informatics, Kyoto University, where he is currently a senior lecturer. His research interests include VLSI design methodology, image processing, and pattern recognition. He is a member of IEEE, IEICE, and IPSJ.

Takashi Sato received B.E. and M.E. degrees from Waseda University, Tokyo, Japan, and a Ph.D. degree from Kyoto University, Kyoto, Japan. He was with Hitachi, Ltd., Tokyo, Japan, from 1991 to 2003, with Renesas Technology Corp., Tokyo, Japan, from 2003 to 2006, and with the Tokyo Institute of Technology, Yokohama, Japan. In 2009, he joined the Graduate School of Informatics, Kyoto University, Kyoto, Japan, where he is currently a professor. He was a visiting industrial fellow at the University of California, Berkeley, from 1998 to 1999. His research interests include CAD for nanometer-scale LSI design, fabrication-aware design methodology, and performance optimization for variation tolerance. Dr. Sato is a member of the IEEE, ACM, and IEICE. He received the Beatrice Winner Award at ISSCC 2000 and the Best Paper Award at ISQED 2003.